**Задачи върху рекурсия**

1. **Глобални променливи:**

const int N = 3; - броят на елементите

public static int[] permutationArray = new int[N]; - масив от тип int, в който ще се съхраняват елементите от 1 до N

public static void GetPermutationArray()

{

for (int i = 0; i < N; i++)

{

permutationArray[i] = i + 1;

}

}

GetPermutationArray() – функция, която добавя в масива всички елементи от 1 до N

1. **Рекурсивен метод:**

Дъното на рекурсията „N“ се достига след „N“ разменяния на елементи от масива и се изписва на екрана цялата пермутация.

public static void RecursionPermutation(int currentIndex)

{

if (currentIndex == N)

{

for (int i = 0; i < N; i++)

{

Console.Write(permutationArray[i]);

}

Console.WriteLine();

}

else

{

for (int i = currentIndex; i < N; i++)

{

Swap(ref permutationArray[currentIndex], ref permutationArray[i]);

RecursionPermutation(currentIndex + 1);

Swap(ref permutationArray[currentIndex], ref permutationArray[i]);

}

}

}

private static void Swap(ref int a, ref int b)

{

int tmp = a;

a = b;

b = tmp;

}

Вторият Swap() в рекурсивната функция се използва, за да върне пермутацията в предишния й вид.

1. **Нерекурсивен метод:**

Първо се създава лист, в който ще се пазят готовите пермутации. В него се добавя първият елемент от масива. С цикъл се обхождат всички елементи от масива, като първоначално се взима текущия елемент от масива с индекс “i” и броят на текущите пермутации. Във втория цикъл се създава нов масив, в който се запазват, създадените до момента пермутации и избраната пермутация се премахва от листа. В третия цикъл се създават пермутациите.

public static void NonRecursionPermutations()

{

List<int[]> permutations = new List<int[]>();

permutations.Add(new int[] { permutationArray[0] });

for (int i = 0; i < N; i++)

{

int currentElement = permutationArray[i];

int currentIndex = permutations.Count;

for (int j = currentIndex - 1; j >= 0; j--)

{

int[] currentPermutation = permutations[j];

permutations.Remove(currentPermutation);

for (int k = 0; k < currentPermutation.Length; k++)

{

permutations.Add(Insert(currentPermutation, k, currentElement));

}

}

}

for (int i = 0; i < permutations.Count; i++)

{

int[] tmpArr = permutations[i];

if (tmpArr.Length > N)

{

tmpArr = tmpArr.Reverse().Skip(1).Reverse().ToArray();

}

permutations[i] = tmpArr;

}

foreach (var item in permutations)

{

for (int i = 0; i < N; i++)

{

Console.Write(item[i]);

}

Console.WriteLine();

}

}

private static int[] Insert(int[] currentPermutation, int position, int currentElement)

{

int[] array = new int[currentPermutation.Length + 1];

for (int i = 0; i < position; i++)

{

array[i] = currentPermutation[i];

}

array[position] = currentElement;

for (int i = position + 1; i < array.Length; i++)

{

array[i] = currentPermutation[i - 1];

}

return array;

}

Методът Insert() служи за добавяне на текущия елемент (currentElement) на позиция (position) и всички елементи надясно от индекс (position) се преместват с една позиция на дясно.

Написаният горе код:

for (int i = 0; i < permutations.Count; i++)

{

int[] tmpArr = permutations[i];

if (tmpArr.Length > N)

{

tmpArr = tmpArr.Reverse().Skip(1).Reverse().ToArray();

}

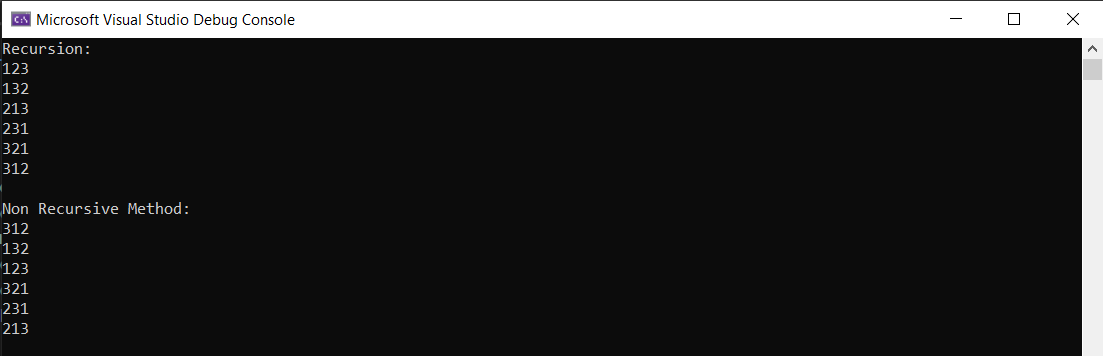
permutations[i] = tmpArr;

}

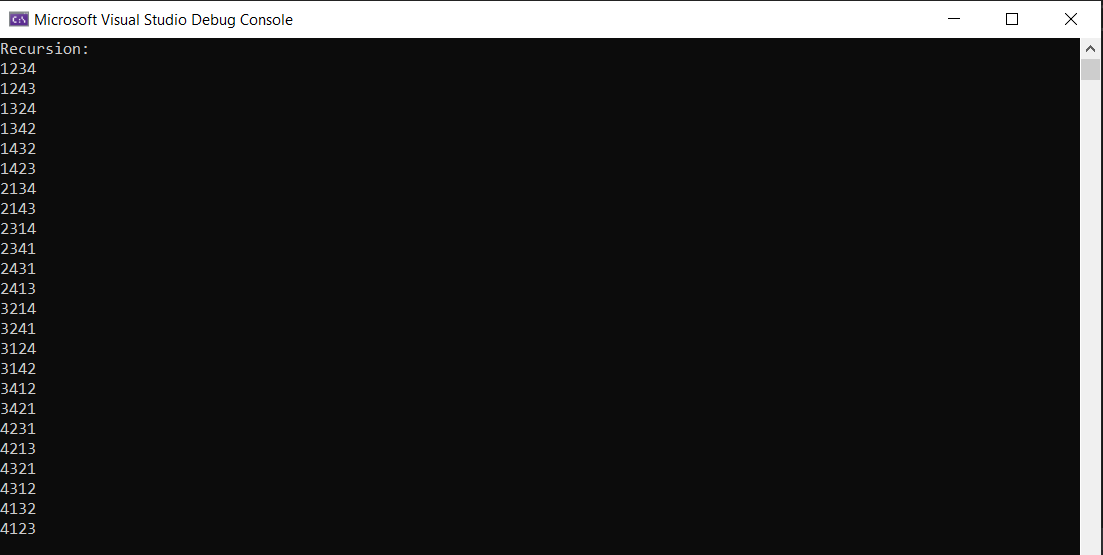
служи за премахване на последния елемент от масивите на пермутациите в листа, тъй като в метода Insert() се добавя един излишен елемент.

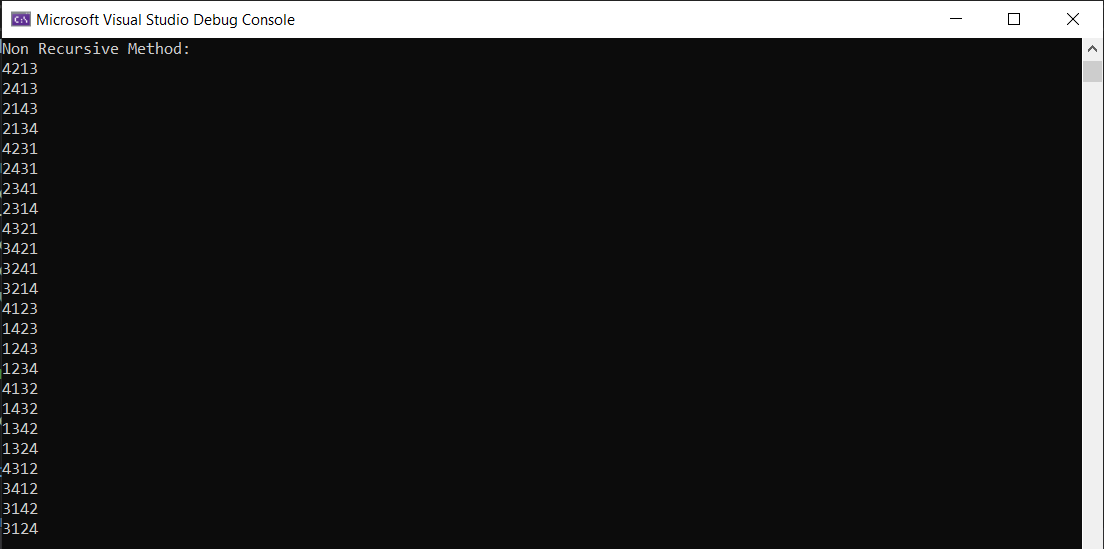
1. **Резултати от двете фунцкии:**

* **N = 3:**

****

* **N = 4:**

****

****